CDF/PUB/XXXX/PUBLIC/XXXX
Draft 0.0
August 23, 2004

Updating X-Mon Fitting to Use Multiple Points Per Run.

Michael Sutherland[1]
Charles Plager[2]
UCLA, USA

Abstract

We updating the fitting code for X-mon to incorporate multiple points per run. This upgrade reduces the number of runs required to get a good fit.

1 Introduction

X-mon, the cross-section monitor for run CDF's run IIa, compares the cross-section of CDF's triggers to know values for different instantaneous luminosities. Currently the fits to get the known values are done by plotting cross-section vs. average instantaneous luminosity [3] for every trigger with one point per run(see figure **??**). The values for these plots are obtained using the "kitchenSink" root file. The new system directly queries the database to get its values and so avoids the problems the root file has been having lately.

2 How the New System Works

The new fitting code for X-mon works by getting multiple points per run to make fits. The data for the cross-sections is obtained by directly querying the database. A Perl script is used to call a series of sql-plus scripts that actually do the database querying. Data is updated approximately every two minutes in the database, but not on the same cycle for the different values. We get around this by binning the data. The cross-sections are then calculated for each bin along with the instantaneous luminosity by the Perl script. All of this is then stored by run number and trigger name. We then updated the old fitting code to read in the data from the files written by the Perl script and fit the data.

[1] sthrlnd(at)fnal.gov
[2] cplager(at)fnal.gov

[3] Average instantaneous luminosity is the geometric average of the instantaneous luminosity at the start and end of the run.

2.1 Querying the Database

All of the information for the fitting code is obtained directly from the database. We wrote sql-plus scripts to get the different trigger counts, luminosities, and trigger table[4]. The different trigger counts are obtained from the LEVEL#_TRIGGER_CONDITIONS where "#" is either 1, 2, or 3, depending on the level of the trigger. All of the available trigger counts are obtained from the database, but as of now, only the live, prescaled counts are used. These counts are updated approximately every two minutes, but the times from each level are a little different. We get similar two minute interval data for the luminosity data, but these times are also different from the trigger count times. We obtain the instantaneous, integrated, and live integrated luminosity data for each run from the BEAM_CONDITIONS part (directory, section?) of the database.

2.2 Making the Bins

We found that all of the times within a level of the triggers had the same two minute cycles but that the different levels and the luminosity data all have slightly different cycles [5]. To increase accuracy, we decided to bin the data at the points where the times lined up closely. Since the sampling rates are all close to each other there is a "beat" to the difference in the data where all of the differences in time get close for multiple points and then slowly get farther apart again. To make the bins we set a thresh hold to find the low sections of the 'beats" and then search the surrounding area for a local minimum. We then use these minimums to define our bins. We construct bins between the luminosity times and each level of the triggers separately. Now each trigger is binned with the luminosity data so that cross-sections can be calculated.

2.3 Calculating the Cross-section

The cross-section for each trigger is calculated from the trigger counts and luminosity data. The live, prescaled trigger counts are used in accordance with the live integrated luminosity to account for deadtime. The cross-section is calculated by:

$$cross\ section = \frac{live\ prescaled\ trigger\ count}{live\ integrated\ luminosity} \tag{1}$$

We then pair this data with the geometric average of every reading of instantaneous luminosity within a bin. These data points are then written to file along with other information about the run.

[4] We want to thank Bill Badgett for all of the help he gave us to figure out how to get the information we needed from the database.

[5] The level 1 and 2 triggers have the same times as well, but we decided to bin them separately with the same algorithm in case that ever changed, or if there was a problem that only effected one of the levels.

2.4 File Structure for Storing the Data Points

After it is called, runQuery.pl will query the database and write all of the information in a folder called "data". If the folder does not exist, it will be created. The file structure used is based on run number, level, and then trigger name. For example, the information for trigger L1_SEVEN_TRK2_v2 for run 185332 would be stored in data/185/185332/L1/L1_SEVEN_TRK2_v2.

2.5 Fitting the Data

The fitting code looks at all the points for a trigger and uses root to find a best fit line for the data. The code uses the equation:

$$f = a + \frac{b}{x} + cx \tag{2}$$

to find the best fit line. Equation (2) is fixed into X-mon to compare the real-time data to the known trends. Our fitting code finds the values of a, b, and c for X-mon for each trigger. To help root find the best fit, the data is preprocessed to get seed values for a, b, and c. Four different fits are tried with the seeds, where some of the values are fixed at zero. There is a flat, constant + $\frac{1}{x}$, linear, and linear + $\frac{1}{x}$. The fit with the best probability is chosen for the trigger. If two fits have the same probability, chi squared (χ^2) and degrees of freedom (DF) is then calculated for the different fits and the one with the lowest value of $\frac{\chi^2}{DF}$ is chosen. An error value is then calculated to see how well the best fit line fits with a minimum value for the error set at 0.20. The values of a, b, c, and the error then get written to a HTML file containing the graphs. All of these values are then picked up later using a Perl script that enters the values into the database for X-mon to use.

3 Using the New Fitting Code

Using the new fitting code consists of two steps.

1. Querying the database for information about specific runs.

2. Using that information to run the fitting code.

We chose to use two steps so that different groups of run numbers could be used at each stage. One would want to only query the database once for each run number but may want to use that run for many different fits of X-mon. Also there might be a mix of old and new run numbers so separating the two processes and saving the run number information to file should reduce the amount of queries to the database and thereby make the fitting code run faster.

3.1 Querying the Database

The Perl script runQuery.pl handles querying the database for the new fitting code. The code uses a class called Trigger which is located in the file Trigger.pm, which accompany runQuery.pl. There should also be a folder in the same directory as runQuery.pl named sqlScripts that contain all of the sql-plus scripts that do the actual database querying. To use runQuery.pl, one first has to run "setup oracle" to enable sql-plus scripts and then run "unsetup perl" to make sure the default version of Perl is still being used. We suggest using "perl runQuery.pl" to make sure the script runs correctly. runQuery.pl has the following options:

```
Usage: runQuery.pl [--option] arguments
-file, --file   get run numbers from file, filename must follow
-f, --force     overwrite data from run numbers whose directories already exist
-h, --help      prints this help page.
```

If no file flag is used the run numbers are taken as the remaining command line arguments. After it is called, runQuery.pl will query the database and write all of the information in a folder called "data". If the folder does not exist, it will be created.

3.2 Running the Fitting Code

The fitting code is contained in the files TriggerLineInfo.cc, TriggerLineInfo.h, Sum.cc, and Sum.h. There are a lot of options when using the fitting code. First one has to decide which mode, Root file or database, they would like to use.

If the Root file is used, the Root file must be linked or copied to the same directory as the fitting code. The file xAlum.C currently calls the file Alum.C which reads in the root file and then calls a series of TriggerLineInfo functions. It is recommended to modify this file to use the Root file mode of the fitting code. For example, Alum.C sets up all of the triggers in an array and then fits them one at time. To change what triggers are fit, one should just change which triggers are processed. The ranges that the graphs will use to color the points are set in the file "runranges.txt". Specific directions are commented at the top of the file.

To use the database mode, the directory data from the runQuery.pl script must be in or linked to the same directory as the fitting code. The database mode is currently set up in the script runDatabaseFits.C. The triggers can be fitted all at once or by only processing those triggers whose names contain a certain string of characters. If the script is called more than once with different sets of run numbers, different triggers can be fitted to different sets of run numbers. The run numbers are automatically obtained from the file "runNumbers.txt", but can be changed to a different file as shown in runDatabaseFits.C.

There are many runs where something went wrong during data taking or that just were not long enough to give a good amount of data. We have two files to help eliminate these runs from the fits:

badsingle.txt contains a list of run numbers that are considered bad.

baddouble.txt contians a list of run ranges where all the run numbers in the ranges are considered bad.

Once bad runs are identified, they can be entered in one of the files above and will not be used in the fitting code, even if they are include in the list of run numbers. To help find bad runs and get better fits, we created a function, TriggerLineInfo::makeSums(), that creates a file called "sums.html" that summarize the runs for the series of fits just processed. "sums.html" analyzes the run numbers over all of the triggers to find those runs that do not fit well for a large number of triggers [6].

[6] This feature is extra helpful when using the Root file mode because all of the runs between the first and last run number of "runranges.txt" that used physics tables are used unless they are in one of the bad run files.